Problem 1
We have a sheet of paper with \( n \) dots, at coordinates \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\). One can fold the paper along a line so that two dots get aligned (in other words, after the folding, they are at the same location). For some lines, more than one pair of dots gets aligned. Design an \( O(n^2 \log n) \) algorithm that computes the maximum number of pairs of dots that can get aligned by folding the paper along a single line in the plane.

A simple example: Suppose we have two dots \((1, 0)\) and \((3, 0)\). We can fold the paper along the vertical line that passes through \( x = 2 \) to align the two dots (i.e., one pair of dots). Another simple example: If we have four dots \((1, 0), (3, 0), (0, 1), \) and \((0, 3)\), we can fold along the line \( x = y \) to make two pairs aligned.

Problem 2
This question is about the Master Theorem.

a) For each of the following recurrences, use the Master theorem to express \( T(n) \) as a Theta of a simple function. State what the corresponding values of \( a, b, \) and \( f(n) \) are and how you determined which case of the theorem applies. Do not worry about the base case or rounding.

1. \( T(n) = T(n/2) + n \)
2. \( T(n) = 4T(n/2) + n \)
3. \( T(n) = 8T(n/2) + n^3 \)

b) Suppose that \( T(n) = T(n/2) + \log n \). Can the Master Theorem be applied to this recurrence? Reason your answer.

Problem 3
Consider the following divide-and-conquer algorithm that assumes a global array \( A \) of integers:

```python
WHATDOIDO(integer left, integer right):
    if left==right:
        if A[left]<0 return (0, 0, 0, A[left])
        else return (A[left], A[left], A[left], A[left])
    if left<right:
        m = (left+right)/2 (rounded down)
        (lmaxsum, llmaxsum, lrmaxsum, lsum) = WHATDOIDO(left, m)
        (rmaxsum, rlmaxsum, rrmaxsum, rsum) = WHATDOIDO(m+1, right)
        maxsum = max{lmaxsum, rmaxsum, lrmaxsum+rlmaxsum}
```

1
leftalignedmaxsum = max{llmaxsum, lsum+rlmaxsum}
rightalignedmaxsum = max{rrmaxsum, lrmmaxsum+rsum}
sum = lsum+rsum
return (maxsum, leftalignedmaxsum, rightalignedmaxsum, sum)

Before running the algorithm, we ask the user to enter \( n \) integers that we store in the array \( A \). Then we run WHATDOIDO(1,n).

a) State the recurrence for \( T(n) \) that captures the running time of the algorithm as closely as possible.

b) Use the “unrolling the recurrence” or the mathematical induction to find a tight bound on \( T(n) \).

c) What does the algorithm do? Specify the input and the output of the algorithm (see the slides for examples).

**Problem 4**

(a) We are given a sequence of \( n \) numbers \( a_0, a_1, \ldots, a_{n-1} \). We would like to determine whether there exists an integer \( x \) that occurs in the sequence more than \( n/2 \) times (i.e., whether the sequence has a majority element). Design an algorithm that runs in time \( O(n) \) and argue its correctness and running time estimate.

Example: For \( A = [3, 1, 2] \) the answer is NO. For \( A = [3, 1, 3] \) the answer is YES.

(b) We are given a sequence of \( n \) numbers \( a_0, a_1, \ldots, a_{n-1} \). We would like to determine whether there exists an integer \( x \) that occurs in the sequence more than \( n/3 \) times. Design an algorithm that runs in time \( O(n) \) and argue its correctness and running time estimate.